VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Porovnávání relačních databází

Comparing Relational Databases

2014 Pavel Hrabovský

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Pavel Hrabovský**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Porovnávání relačních databází
Comparing Relational Databases

Zásady pro vypracování:

Cílem práce je navrhnout a naimplementovat aplikaci s grafickým uživatelským rozhraním, která bude sloužit k porovnání dvou relačních databází s objektově-relačním rozšířením a následně pro generování opravných SQL skriptů. Práce může mít praktické využití pro oddělený vývoj na testovací databázi, kdy je potřeba opravené skripty generovat po dokončení vývojového cyklu.

Práce bude splňovat následující body:
1. Rozbor objektů (relací, pohledů, uložených procedur atd.) ve zvoleném relačním SŘBD s objektově-relačním rozšířením.
2. Popis možností systémového katalogu.
3. Implementace aplikace pro porovnání dvou databází. Porovnávána budou relační schémata (nově vzniklé relace, odstraněné relace, nově vzniklé atributy atd.) a prvky objektově-relačního rozšíření (nově vzniklé funkce, upravené funkce atd.).
4. Rozšíření aplikace o generování opravných SQL skriptů.
5. Srovnání s existujícími aplikacemi.

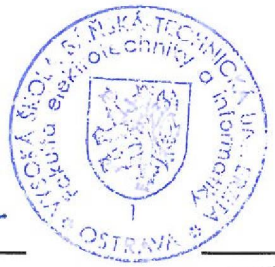
Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Lukáš**

Datum zadání: 01.09.2013
Datum odevzdání: 07.05.2014



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 16. dubna 2014 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, především panu Ing. Petru Lukášovi, protože bez nich by tato práce nevznikla.

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací aplikace s grafickým uživatelským rozhraním pro porovnání schémat dvou relačních databází s objektově relačním rozšířením. Práce popisuje architekturu databáze a systémový katalog vybraného SŘBD (systému řízení báze dat) a jak je možné tohoto katalogu využít pro sledování změn. Výsledek práce může mít praktické využití pro oddělený vývoj na testovací databázi, kdy je nutné po ukončení vývojového cyklu přenést změny na reálnou databázi v provozu.

Klíčová slova: porovnání relačních databází, systémový katalog, SQL, T-SQL

Abstract

This bachelor thesis deals with a proposal and implementation of an application with graphical user interface for comparing two schemas of relational databases with object-relational extension. The work describes an architecture and a system catalog of a chosen DBMS (Database Management System) and how the catalog can be used for tracking changes. The result of this thesis can be used in practice when we need to apply changes, which we have made on testing database, to a real database.

Keywords: comparing relational databases, system catalog, SQL, T-SQL

Seznam použitých zkratek a symbolů

ID	–	Identifikační číslo
SQL	–	Structured Query Language (strukturovaný dotazovací jazyk)
SQL - 92	–	Třetí revize strukturovaného dotazovacího jazyka SQL
SŘBD	–	Systém řízení báze dat
T-SQL	–	Transact-SQL
Trial verze	–	Časově omezená verze programu

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

1.1 Úvod

Cílem tohoto textu je seznámit čtenáře s objekty ve zvoleném SŘBD s objektově relačním rozšířením – zejména se budeme věnovat objektům, které jsou důležité z hlediska porovnání relačních databází. Dále si popíšeme možnosti práce se systémovým katalogem. Práce se dále zabývá popisem aplikace pro porovnání dvou relačních databází a postupem při implementaci této aplikace. Aplikace vytvořená v této práci může být využita pro oddělený vývoj na testovací databázi, kdy je potřeba opravné skripty generovat po dokončení vývojového cyklu.

V kapitole 2 se budeme věnovat popisu struktury databáze a rozebereme si objekty, které jsou v ní obsaženy. Následující kapitolu 3 budeme věnovat jazyku T-SQL a popisu jeho příkazů. Rozebereme zejména ty příkazy, jež slouží pro vytváření, modifikaci nebo mazání objektů. V další kapitole 4 se budeme zabývat nejvhodnějším přístupem do systémového katalogu – systémovými pohledy, které si zde popíšeme. V nadcházející kapitole 5 si popíšeme ovládání vytvořené aplikace a postup při její implementaci. V další kapitole 6 si tuto aplikaci srovnáme s již existujícími aplikacemi. V poslední kapitole 7 si celou práci shrneme a uzavřeme.

1.2 Volba SŘBD

Ze všeho nejdříve si musíme ujasnit, který SŘBD s objektově – relačním rozšířením budeme popisovat. Můžeme nalézt pestrou škálu SŘBD, které mezi sebou mají větší či menší rozdíly. Mezi jednotlivé zástupce patří např. DB2 od firmy IBM, Oracle Database, či Microsoft SQL Server. V této práci se budeme zabývat SŘBD od firmy Microsoft, konkrétně SŘDB SQL Server – verze 2012, jeho jazykem SQL s rozšířením T-SQL.

2 Architektura databáze

V této kapitole si popíšeme objekty systémového katalogu Microsoft SQL Serveru, které jsou důležité z hlediska porovnání databází, tyto objekty poté budeme porovnávat v samotné aplikaci.

2.1 Objekty obsažené v databázi MS SQL Serveru

- tabulky, pohledy, synonyma
- sloupce, indexy
- omezení – primární klíče, cizí klíče, kontrolní omezení, výchozí omezení
- triggery
- procedury a funkce
- databázové principy – uživatelé, role, oprávnění
- datové typy
- agregační funkce
- sekvence
- vztahy a dědičnosti

2.1.1 Schémata (Schemas)

Databázové schéma je kolekce databázových objektů (tabulek, procedur, triggerů atd.). V databázi je několik výchozích schémat. Nejdůležitějším schématem pro tuto práci je schéma `dbo`. Toto schéma je použito jako výchozí pro ukládání uživatelem definovaných objektů. Aplikace vytvořená v této práci bude porovnávat data vybraná právě ze schématu `dbo`. Dalším důležitým schématem je `sys`, ze kterého budeme spouštět systémové pohledy. Více o systémových pohledech v kapitole 4.

2.1.2 Tabulky (Tables)

Základním pojmem relačních datových modelů je relace – neformálně prezentována jako tabulka. V relační databázi jsou datové soubory chápány jako množiny (viz definice 2.1). Z pohledu uživatele jsou data v relačním datovém modelu uspořádána v dvourozměrných tabulkách, kdy je každá z těchto tabulek označována pojmem relace. Tabulky jsou základní databázové objekty tvořené atributy (sloupci). Každý záznam (řádek) v této tabulce poté představuje nějaký prvek reálného světa. Průsečíky záznamu a atributu nazýváme pole.

- Sloupce (Columns) – Seznam atributů (sloupců) patří k definici databázové tabulky. U atributu se udává název, datový typ, rozsah.
- Řádky (Rows) – Každý záznam (řádek) tabulky reprezentuje individuální zastoupení objektu (např. každého zaměstnance ve firmě).

Definice 2.1 *(Relace). Nechť $D_1, D_2, ..., D_n$ jsou množiny. N-ární relaci na množinách $D_1, D_2, ..., D_n$ nazýváme libovolnou konečnou podmnožinou kartézkého součinu, $R \subseteq D_1 \times D_2 \times ...D_n$.*

Ke každé tabulce se rovněž vztahují integritní omezení, což jsou omezení která ošetřují vstup, nebo výstup dat z databáze. Více si jednotlivé typy integritních omezení popíšeme v podkapitole 2.1.5.

2.1.3 Pohledy (Views)

Jedná se o databázové objekty, které poskytují uživateli data ve stejné podobě jako tabulka. Samotný pohled neobsahuje data, ale pouze předpis, který říká, jakým způsobem jsou data získávána z tabulek a jiných pohledů.

Pohled, na rozdíl od tabulky, nezabírá téměř žádné místo v paměti databáze, protože neobsahuje data, ale pouze předpis k jejich získání. Pohledy i tabulky lze používat v SQL dotazech.

2.1.4 Indexy (Indexes)

Index je libovolná datová struktura s lepší než lineární složitostí vyhledávání. Jako příklad této struktury můžeme uvést například B-strom. Pomocí indexů lze přistupovat k datům rychleji, než za použití sekvenčního vyhledávání. Index je definován tabulkou a výběrem jednoho nebo více konkrétních sloupců.

2.1.5 Omezení (Constraints)

Jedná se o dodatečná integritní omezení atributů (sloupců) tabulky.

Primární klíče (Primary Keys) Primární klíč je jednoznačný identifikátor řádku v tabulce. Může být tvořen sloupcem, či kombinací sloupců a to tak, aby byla zaručena jeho jednoznačnost. Pole s klíčem musí obsahovat nějakou hodnotu, tzn. nesmí se zde vyskytovat prázdná nedefinovaná hodnota `NULL`. V praxi se často používají „umělé“ primární klíče, což mohou být číselné, či písemné identifikátory. Každý záznam zde dostává unikátní identifikátor, který je odlišný od identifikátorů předchozích záznamů. Většinou se jedná o celočíselné řady, kdy každý nový záznam dostává identifikátor s číslem o jednotku vyšší, než záznam předchozí. Toto je většinou prováděno zcela automaticky.

Cizí klíče (Foreign Keys) Taktéž nazývané jako nevlastní klíče. Tyto klíče slouží pro vyjádření vztahů mezi databázovými tabulkami. Cizí klíč je složen z jednoho či více polí. Tato pole nám umožňují identifikovat, které záznamy z různých tabulek spolu navzájem souvisejí.

Kontrolní omezení (Check Constraints) Tímto omezením zajišťujeme integritu dat a to tak, že omezíme hodnoty, které je možné zadat pro jeden, nebo více sloupců. Kontrolu omezení můžeme vytvořit s jakýmkoli logickým (boolean) výrazem, jehož návratová hodnota je `TRUE`, nebo `FALSE`. Jako příklad můžeme uvést rozsah hodnot pro plat, který chceme mít pouze v rozsahu od 15 000 Kč do 100 000 Kč. Toto omezení zamezí vložení dat, která nejsou v daném rozsahu. Logický výraz pro toto omezení bude následující: `plat >= 15000 AND plat <= 100000`.

Na jeden sloupec lze samozřejmě aplikovat více kontrolních omezení, taktéž lze toto omezení aplikovat nad tabulkou a tím omezit více sloupců najednou.

Kontrolní omezení jsou podobná cizím klíčům – omezují hodnoty, které lze do sloupce zadat. Rozdíl mezi nimi je ten, že cizí klíče vybírají validní hodnoty z jiné tabulky, zatímco kontrolní omezení tyto hodnoty vybírají z logického výrazu, kterými jsou definovány.

Kontrolním omezením neprojdou hodnoty, které jsou dle logického výrazu vyhodnoceny jako `FALSE`.

Výchozí omezení (Default Constraints) Atribut v tabulce může být definován společně s jeho výchozí hodnotou (Default Constraint). Při vkládání záznamu do tabulky se velmi často neuvádí hodnoty všech atributů nového záznamu. Hodnoty neuvedených atributů se do nového záznamu uloží jako `NULL` nebo pokud je definována výchozí hodnota, použije se tato hodnota.

2.1.6 Triggery (Triggers)

Triggery jsou speciální druh uložených procedur, definují činnost, která se má provést při definované události nad danou tabulkou. Tato událost může být například vložení nebo smazání dat.

Microsoft SQL Server rozlišuje 2 druhy triggerů:

- DML Triggery – tyto jsou prováděny při vykonání DML události, tyto události jsou `INSERT`, `UPDATE`, `DELETE`. Tyto triggery můžou být použity k zavedení různých omezení např. můžou hlídat formát zadávaných dat. Existují tyto typy DML triggerů:
  - AFTER Triggery – jsou spouštěny po vložení, aktualizaci nebo smazání záznamu. Trigger se nevykoná v případě, že by porušil omezení, která jsou již definována v databázi.

  - INSTEAD OF Triggery – přepisují standardní akce, tudíž mohou být využity pro ošetření chyb nebo ke kontrole zadávané hodnoty. Jsou vykonávány před vložením, aktualizací nebo smazáním řádku.

- DDL Triggery – reagují na řadu údajů DDL událostí. Tyto události jsou převážně příkazy `CREATE TABLE`, `ALTER TABLE a DROP TABLE` jazyka T-SQL. Taktéž mohou reagovat na různé systémové uložené procedury, které provádějí DDL operace. Tyto triggery mohou být vytvořeny přímo z T-SQL nebo z metod vytvořených pomocí Microsoft .NET Frameworku.

2.1.7 Uživatelem definované procedury a funkce

Uživatelem definované procedury a funkce jsou pojmenované programové bloky, ve kterých se kombinuje použití standardních SQL příkazů pro dotazování a práci s daty s programovými konstrukcemi jako jsou například podmínky a cykly. Tyto bloky lze poté jednoduše zavolat tak, že za příkaz `EXECUTE` uvedeme název uložené procedury či funkce. Procedurám i funkcím můžeme předávat vstupní parametry. Procedury a funkce se liší v tom, že funkce mají nějakou návratovou hodnotu. Funkce můžeme na SQL Serveru dělit do dvou kategorií.

- Skalární funkce (Scalar Functions). - Návratová hodnota těchto funkcí je jediná skalární hodnota.

- Table – Valued funkce (Table Valued Functions) - Tyto funkce vracejí tabulku tvořenou atributy, které jsou deklarovány uvnitř funkce. Tyto funkce se dále dělí na ty, jež obsahují nějaký programový kód, který nakonec vytvoří výslednou tabulku – tyto se nazývají „Multi Statement Table – Valued funkce“, a na funkce, které obsahují jediný SQL dotaz – „Inline funkce“. Tyto funkce se mohou využít jako alternativa k pohledům. Jejich výhoda spočívá v možnosti využít více `SELECT` příkazů, na rozdíl od pohledů, které jsou limitovány použitím pouze jednoho příkazu `SELECT`.

2.2 Datové typy (Data Types)

Objekty v databázi, které obsahují data mají asociovaný určitý datový typ. V následujícím seznamu můžeme vidět, které objekty mají datové typy:

- Sloupce tabulek a pohledů.

- Parametry uložených procedur a funkcí.

- Lokální proměnné programových bloků.

- T-SQL funkce, které vracejí jednu nebo více datových hodnot specifického datového typu.
- Návratové datové typy skalárních funkcí.

Přiřazení datového typu danému objektu definuje následující čtyři atributy tohoto objektu:

- Typ dat obsažených v objektu.
- Délku nebo velikost ukládané hodnoty např. u řetězců.
- Přesnost čísla (počet desetinných míst, pouze u číselných hodnot).
- Měřítko čísla (pouze u číselných hodnot).

Dále si popíšeme jednotlivé datové typy obsažené v T-SQL.

Přesné číselné typy Jedná se o datové typy, které definují rozsah a přesnost. Jsou vhodné pro operace s čísly, kde nesmí dojít ke ztrátě přesnosti např. finanční operace. Nejvíce využívaný číselný datový typ v databázi je `int`. Datový typ `bigint` se používá v případě, že je `int` nedostatečný.

Datový typ	Informace	Paměťový prostor
bigint	Rozsah: $-2^{63}(-9223372036854775808)$ až $2^{63-1}(9223372036854775807)$	8 B
int	Rozsah: $-2^{31}(-2147483648)$ až $2^{31-1}(2147483647)$	4 B
smallint	Rozsah: $-2^{15}(-32768)$ až $2^{15-1}(32767)$	2 B
tinyint	Rozsah: 0 až 255	1 B
decimal (p,s)	Číslo s desetinnou čárkou, kde p je maximum cifer a s je počet desetinných míst. V paměti zabírá 5 až 7 bytů podle rozsahu. Pokud rozsah neuvedeme, bude se jednat o ekvivalent decimal (18,0)	5 - 7 B
numeric (p,s)	Ekvivalent pro decimal	5 - 7 B
smallmoney	Desetinné číslo uváděné s přesností na deseti tisíciny. Rozsah: $-214748,3648$ až $214748,3647$	4 B
money	Přesnost je stejná jako u smallmoney. Rozsah: $-922337203685477,5808$ až $922337203685477,5807$	8 B

Tabulka 1: Celočíselné datové typy

Přibližné číselné typy Tyto datové typy mají velký rozsah, ale na úkor přesnosti. Čím větší číslo bude, tím se bude desetinná čárka více posouvat a tím je větší šance ztráty přesnosti na nižších řádech čísla. Tento typ je vhodný pro použití tam, kde provádíme operace s tolerancí k výpočtu.

- float (*n*) – Číslo s plovoucí desetinnou čárkou, kde *n* může být v rozsahu 1 - 53 a určuje počet bitů. Přesnost čísla je zde přímo ovlivněná celkovým počtem cifer. Celkové nároky na paměť u tohoto datového typu jsou 4 nebo 8 bytů.
- real – Odpovídá datovému typu float, pokud za *n* dosadíme hodnotu 24.

Binární Binární datové typy mohou být, buď s pevnou nebo proměnnou délkou. Slouží pro ukládání binárních dat, jako jsou například soubory a obrázky.

- binary[(*n*)] – Binární datový typ s pevnou délkou s velikostí *n* bajtů, kde *n* je hodnota v rozsahu od 1 do 8000. Tento datový typ zabírá v paměti *n* bajtů.
- varbinary[(*n* | `max`)] – Typ s proměnnou délkou, hodnota n může nabývat stejných hodnot jako v předchozím případě. Slovo `max` nám udává, že maximální úložná kapacita tohoto datového typu je 2^{31-1} bajtů, nicméně, nároky na paměť odpovídají velikosti vložených dat + 2 bajty.

Bit Bit je nejmenší jednotka informace. Jeho hodnota může být 1, nebo 0. Tyto dvě hodnoty představují logické hodnoty `TRUE`, nebo `FALSE`. SQL server optimalizuje potřebnou velikost paměti pro sloupce, jejichž datový typ je `bit`. Pokud je takovýchto sloupců v tabulce osm a méně, jsou sloupce uloženy jako 1 bajt. Pokud je sloupců od 9 až po 16 jsou uloženy jako 2 bajty a takto můžeme pokračovat dále.

Hodnoty textových řetězců 'TRUE' a 'FALSE' mohou být konvertovány do hodnoty typu `bit`. Hodnota 'TRUE' bude konvertována do 1 a hodnota 'FALSE' do 0.

Text Mohou být pevné, nebo proměnné délky.

- char [(*n*)] – Textový řetězec s pevnou délkou, který není UNICODE. Hodnota *n* může nabývat rozsahu od 1 do 8000. Tento datový typ zabírá v paměti *n* bajtů.
- varchar [(*n* | max)] – Textový řetězec s proměnnou délkou. Stejně jako v předchozím případě může *n* nabývat hodnot od 1 do 8000. V případě použití slova `max` zabírá hodnota v paměti velikost 2^{31-1} bajtů, jinak nároky na paměť odpovídají velikosti vložených dat + 2 bajty.

Datum, čas Tyto datové typy definují datum, popřípadě datum s časem.

- date – definuje pouze datum

Vlastnost	Hodnota
Syntaxe	date
Výchozí formát řetězce	YYYY-MM-DD (ROK-MĚSÍC-DEN)
Rozsah	od 0001-01-01 do 9999-12-31
Délka řetězce	10 pozic
Nároky na paměť	Pevné, 3 bajty
Přesnost	Jeden den
Výchozí hodnota	1900-01-01
Kalendář	Gregoriánský

Tabulka 2: Popis datového typu – date

- datetime – Definuje datum kombinovaný s časem se sekundami. Čas je založen na dvaceti čtyř hodinovém formátu.

Vlastnost	Hodnota
Syntaxe	datetime
Rozsah data	od Leden 1, 1753, do Prosinec 31, 9999
Rozsah času	od 00:00:00 do 23:59:59.997
Velikost řetězce	minimum 19 pozic, maximum 23 pozic
Nároky na paměť	8 bajtů
Přesnost	Zaokrouhleno na ,.000, ,.003, .007 vteřin
Výchozí hodnota	1900-01-01 00:00:00
Kalendář	Gregoriánský

Tabulka 3: Popis datového typu - datetime

- datetime2 – Vylepšení předchozího typu. Má větší rozsah a větší přesnost, taktéž u něj lze definovat vlastní přesnost.
- datetimeoffset – Předchozí datové typy neměly povědomí o časových zónách, kdežto tento datový typ má.

3 Jazyk T - SQL

3.1 Popis

V této kapitole provedeme rozbor základních rysů jazyka SQL (Structured Query Language) a jazyka T-SQL, který rozšiřuje SQL o procedurální rysy. T-SQL byl vyvinut firmou Sybase a Microsoft pro SQL Server.

3.2 Popis syntaxe

Syntaxi vybraných příkazů budeme popisovat dle následující notace: klíčová slova příkazů budeme psát velkými písmeny. Do závorek „< >“ zapíšeme syntaktické konstrukce, místo těchto závorek doplníme konkrétní tvary. Nepovinné části příkazů uzavřeme do hranatých závorek „[ ]“. Svislou čarou „|“ oddělíme jednotlivé volitelné varianty, jsou-li varianty uzavřeny ve složených závorkách „{ }“, musí být jedna z těchto variant použita. Třemi tečkami s písmenem n „...n“ vyjádříme, že se konstrukce v předchozí závorce může opakovat.

3.3 Příkazy SQL

SQL příkazy můžeme rozdělit do následujících skupin:

- Příkazy pro dotazování (`SELECT`)
  - `SELECT` tento příkaz nám slouží k výběru záznamů z tabulky. Jedná se zřejmě o nejpoužívanější příkaz jazyka SQL. V tomto příkazu je nutné určit z jaké tabulky (tabulek) je proveden výběr. Taktéž musíme uvést, jaké sloupce mají být do výběru zahrnuty. K tomuto příkazu můžeme uvést i několik nepovinných podmínek, dle kterých lze specifikovat, které záznamy budou vybrány. Celý výstup lze taktéž seřadit vzestupně i sestupně.
- Příkazy pro manipulaci s daty (`INSERT, UPDATE, DELETE`)
  - `INSERT` tohoto příkazu můžeme využít pro vkládání záznamu do tabulky. U tohoto příkazu je třeba uvést do jaké tabulky bude záznam vkládán. Dále můžeme uvést do jakých atributů (sloupců) mají být uloženy dané hodnoty. Při vkládání dat musíme dodržovat datový typ atributů podle schématu tabulky.
  - `UPDATE` za pomoci tohoto příkazu lze modifikovat již existující záznam v tabulce. U toho příkazu je nutno uvést v jaké tabulce bude záznam modifikován a podmínku určující, které záznamy mají být modifikovány.

  - `DELETE` s tímto příkazem lze odebrat zvolený záznam z tabulky. Podobně jako u předchozího příkazu je třeba uvést jméno tabulky, ze které bude záznam smazán a to za uvedených podmínek.

- Příkazy pro definici dat (`CREATE, ALTER, DROP`) - tyto příkazy si rozebereme podrobněji jelikož jsou z hlediska této práce velmi důležité – zejména při generování opravného SQL skriptu v aplikaci.

3.4 Příkazy T-SQL pro definici dat

V této podkapitole se zaměříme na popis syntaxí příkazů pro definici dat. Uváděné popisy syntaxí nejsou kompletní, nicméně pro účely této práce plně postačují.

3.4.1 Tabulky

Vytvoření tabulky Pro vytvoření tabulky použijeme příkaz `CREATE TABLE`. Tabulka se tvoří podle výpisu 1, kde můžeme vidět, že je třeba definovat název tabulky, jednotlivé sloupce tabulky a omezení. Volitelně zadáváme název databáze a název schématu.

```
CREATE TABLE
    [ nazev_databaze . [ nazev_schematu. ] ] nazev_tabulky
    ( { <definice_sloupce> | [ <omezeni_tabulky> ] } [ ,... n ] ) [ ; ]
```

Výpis 1: Syntaxe – Vytvoření tabulky

Sloupce v tabulce definujeme podle syntaxe uvedené na výpisu 2, která nám říká, že je potřeba definovat název každého sloupce, specifikovat datový typ a určit zda může atribut nabývat hodnoty `NULL`. U určitých datových typů je nutno definovat rozsah nebo přesnost. Uvedeme-li místo konkrétního rozsahu klíčové slovo `max`, vytvoří se atribut o maximální velikosti tohoto typu. Klíčovým slovem `CONSTRAINT` můžeme přidat omezení pro vytvářený sloupec, kdy za klíčové slovo `DEFAULT` uvedeme výchozí hodnotu nebo můžeme použít slovo `IDENTITY`, pomocí kterého říkáme, že se má hodnota atributu při vkládání záznamu automaticky zvyšovat, a to od dané startovací hodnoty po nastavených krocích.

```
<definice_sloupce> ::=
nazev_sloupce datovy_typ [ ( delka [ , presnost ] | max |) ] [ NULL | NOT NULL ]
    [ [ CONSTRAINT nazev_omezeni ] DEFAULT omezujici_vyraz ] |
      [ IDENTITY [ ( startovaci_hodnota ,krok_o_ktery_se_ma_zvysit ) ] ]
    [ <omezeni_sloupce> [ ...n ] ]
```

Výpis 2: Syntaxe – Definice sloupců

Ke sloupci můžeme taktéž přidat různá omezení klíčovým slovem `CONSTRAINT`, které můžeme nadefinovat jako primární (`PRIMARY KEY`) či cizí klíč (`FOREIGN KEY`)

nebo jako unikátní hodnotu `UNIQUE`. V případě že jej definujeme jako primární klíče, bude sloupec v tabulce unikátní (tzn. nebude možné mít v tabulce dva záznamy se stejným primárním klíčem). Pokud sloupec definujeme jako cizí klíč, je potřeba se odkazovat na tabulku, ze které tento klíč pochází (v odkazované tabulce je to primární klíč). Toto provedeme klíčovým slovem `REFERENCES`. Musíme se také odkázat přímo na sloupec. Další možností je přidat sloupci kontrolní omezení, a to klíčovým slovem `CHECK`, za kterým je potřeba uvést logický výraz, který se bude kontrolovat. Další informace o těchto omezeních si uvedeme v kapitole 2.

```
<omezeni_sloupce> ::=
[ CONSTRAINT nazev_omezeni ]
{      { PRIMARY KEY | UNIQUE }

  | [ FOREIGN KEY ]
        REFERENCES [ nazev_schematu . ] odkazovana_tabulka [ ( odkazovany_sloupec ) ]
  | CHECK ( logicky_vyraz )
}
```

Výpis 3: Syntaxe – Omezení sloupců

Omezení můžeme také přidávat až za definici sloupců. Toto se provádí podobným způsobem jako v předchozím případě, ale je potřeba specifikovat pro jaký sloupec toto omezení platí. Syntaxi pro zápis můžeme vidět na výpisu 4

```
< omezeni_tabulky > ::=
[ CONSTRAINT nazev_omezeni ]
{
    { PRIMARY KEY | UNIQUE }
        (column [ ,... n ] )

    | FOREIGN KEY
        ( column [ ,... n ] )
        REFERENCES referenced_table_name [ ( ref_column [ ,...n ] )

    | CHECK ( logicky_vyraz )
}
```

Výpis 4: Syntaxe – Omezení tabulky

Modifikace tabulky Příkaz `ALTER TABLE` nám slouží k modifikování již existující tabulky. Umožňuje nám v tabulce přidávat nebo odebírat jednotlivé atributy a měnit jejich datové typy. V případě modifikace tabulky se musíme řídit syntaxí ve výpisu 5, ve kterém vidíme, že je potřeba uvést název modifikované tabulky a poté zvolit, jak se bude tabulka měnit. Můžeme upravovat, přidávat, či odebírat atributy (sloupce) (syntaxe definice sloupce viz výpis 2). Atributy upravujeme klíčovými slovy `ALTER COLUMN`, za které

uvedeme název atributu a příslušné úpravy. Za klíčové slovo `ADD` lze přidávat jednotlivé sloupce (viz výpis 2) nebo omezení (viz výpis 4).

Atributy a omezení lze rovněž smazat a to tak, že se za klíčové slovo `DROP` uvede, zda se má odstranit omezení nebo atribut (`CONSTRAINT` nebo `COLUMN`) a za ně se uvede jejich název. V neposlední řadě můžeme vypínat nebo zapínat triggery nad danou tabulkou, což provedeme klíčovými slovy `ENABLE` nebo `DISABLE TRIGGER`. Za tato klíčová slova poté uvádíme názvy jednotlivých triggerů, popřípadě můžeme uvést klíčové slovo `ALL`, čímž zapneme nebo vypneme všechny triggery nad danou tabulkou.

```
ALTER TABLE [ nazev_databaze . [ nazev_schematu ]. ] nazev_tabulky
{
    ALTER COLUMN column_name
    {
        datovy_typ [ ( { delka [ , presnost ]| max |} ) ] [ NULL | NOT NULL ]
    }

    | ADD

    {
        <definice_sloupce>
      | <omezeni_tabulky>
    } [ ,... n ]

    | DROP

    {
        [ CONSTRAINT ] nazev_omezeni
        | COLUMN nazev_sloupce
    } [ ,... n ]

    | [ WITH { CHECK | NOCHECK } ] { CHECK | NOCHECK } CONSTRAINT
        { ALL | nazev_omezeni [ ,...n ] }

    | { ENABLE | DISABLE } TRIGGER
        { ALL | nazev_triggeru [ ,... n ] }

}[ ; ]
```

Výpis 5: Syntaxe – Úprava tabulky

Smazání tabulky Pomocí příkazu `DROP TABLE` smažeme zamýšlenou tabulku z databáze. Syntaxe příkazu pro smazání tabulky je celkem jednoduchá – stačí definovat název tabulky, volitelně název databáze a název schématu.

```
DROP TABLE [ nazev_databaze . [ nazev_schematu ]. ] nazev_tabulky [ ,...n ] [ ; ]
```

Výpis 6: Syntaxe – Smazání tabulky

3.4.2 Indexy

Vytvoření indexu Index se vytvoří podle syntaxe ve výpisu 7. Nejprve uvedeme klíčová slova `CREATE INDEX`, za která definujeme název indexu. Mezi klíčová slova `CREATE` a `INDEX` můžeme uvést klíčové slovo `UNIQUE`, pomocí kterého zaručíme, že indexované hodnoty budou pro každý záznam jedinečné (podobně jako v případě primárního klíče). Za slovo `ON` dále definujeme na jaký objekt se bude index vztahovat a do závorky uvedeme sloupce na které se bude index vztahovat.

```
CREATE [ UNIQUE ] INDEX nazev_indexu
    ON <object> ( column [ ,...n ] ) [ ; ]

      <object> ::=
{
    [ nazev_databaze. [ nazev_schematu. ]
    nazev_tabulky_nebo_pohledu
}
```

Výpis 7: Syntaxe – Vytvoření indexu

Smazání indexu Index smažeme tak, že za příkaz `DROP INDEX` uvedeme název vytvořeného indexu a za klíčové slovo `ON` uvedeme název tabulky nebo pohledu nad kterými byl index vytvořen. Syntaxe příkazu je uvedena ve výpisu 8.

```
DROP INDEX nazev_indexu ON nazev_tabulky_nebo_pohledu
```

Výpis 8: Syntaxe – Smazání indexu

3.4.3 Triggery

Vytvoření triggeru Trigger vytvoříme za pomocí příkazu `CREATE TRIGGER`. Zde je třeba definovat název triggeru. Dále je potřeba za klíčové slovo `ON` specifikovat název tabulky nebo pohledu, nad kterými se bude trigger vykonávat a za jakých okolností. V neposlední řadě také musíme uvést jednotlivé SQL příkazy. Volitelnou částí je specifikování názvu schématu.

```
CREATE TRIGGER [ nazev_schematu . ]nazev_triggeru
ON { nazev_tabulky | nazev_pohledu }
{ FOR | AFTER | INSTEAD OF }
```

```
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS { sql_prikazy [ ; ] [ ,... n ] } [ ; ]
```

Výpis 9: Syntaxe – Vytvoření triggeru

Modifikace triggeru Modifikace triggeru se provádí za pomoci příkazu `ALTER TRIGGER`, jehož syntaxe je velmi podobná jako v předchozím případě 9 u vytváření triggeru, s tím rozdílem, že místo klíčového slova `CREATE` uvedeme slovo `ALTER`. Tato syntaxe je uvedena na výpisu 10.

```
ALTER TRIGGER [ nazev_schematu . ]nazev_triggeru
ON { nazev_tabulky | nazev_pohledu }
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS { sql_prikazy [ ; ] [ ,... n ] } [ ; ]
```

Výpis 10: Syntaxe – Vytvoření triggeru

Smazání triggeru Trigger smažeme příkazem `DROP TRIGGER`, jehož syntaxe je opět velmi jednoduchá, stejně jako v případě tabulky (viz výpis 6), zaměníme pouze slovo `TABLE` za `TRIGGER`. Syntaxi pro zápis můžeme vidět na výpisu 11.

```
DROP TRIGGER [ nazev_databaze . [ nazev_schematu ]. ] nazev_triggeru [ ,...n ] [ ; ]
```

Výpis 11: Syntaxe – Smazání triggeru

3.4.4 Pohledy

Vytvoření pohledu Vytvoření pohledu provedeme příkazem `CREATE VIEW`, a to dle syntaxe ve výpisu 12. Jedná se v podstatě pouze o zapouzdřený, pojmenovaný příkaz `SELECT`, který se uvádí za klíčové slovo `AS`.

```
CREATE VIEW [ nazev_schematu . ] nazev_pohledu [ (column [ ,...n ] ) ]
AS select_statement[ ; ]
```

Výpis 12: Syntaxe – Vytvoření pohledu

Modifikace pohledu Modifikace se provádí příkazem `ALTER VIEW`, jehož syntaxe je podobná jako v případě vytváření pohledu 12, pouze se zamění příkaz pro vytvoření za příkaz pro modifikaci, syntaxe je uvedena na výpisu 13.

```
ALTER VIEW [ nazev_schematu . ] nazev_pohledu [ (column [ ,...n ] ) ]
AS select_statement[ ; ]
```

Výpis 13: Syntaxe – Modifikace pohledu

Smazání pohledu Syntaxe opět téměř totožná jako u smazání tabulky, zaměníme pouze klíčové slovo `TABLE` za slovo `VIEW`, syntaxi pro tuto změnu můžeme nalézt na výpisu 14.

```
DROP VIEW [ nazev_databaze . [ nazev_schematu ]. ] nazev_pohledu [ ,...n ] [ ; ]
```

Výpis 14: Syntaxe – Smazání pohledu

3.4.5 Procedury

Vytvoření procedury Za pomoci příkazu `CREATE PROCEDURE` (zkráceně `CREATE PROC`) lze vytvořit uloženou proceduru. U tohoto příkazu je potřeba specifikovat název procedury, parametry a taktéž „programový kód“, který se skládá ze standardních SQL příkazů a řídících konstrukcí jako jsou podmínky a cykly, které se budou v této proceduře vykonávat.

```
CREATE { PROC | PROCEDURE } [nazev_schematu.] nazev_procedury
[ { @parameter [ schema_datoveho_typu. ] datovy_typ } ] [ ,... n ]
AS{[ BEGIN ]
sql_prikaz [;] [ ... n ]
[ END ]}[;]
```

Výpis 15: Syntaxe – Vytvoření procedury

Modifikace procedury Syntaxe příkazu je téměř totožná s předchozím případem, musí se pouze zaměnit klíčové slovo `CREATE` za slovo `ALTER`. Syntaxe viz výpis 16.

```
CREATE { PROC | PROCEDURE } [nazev_schematu.] nazev_procedury
[ { @parameter [ schema_datoveho_typu. ] datovy_typ } ] [ ,... n ]
AS{[ BEGIN ]
sql_prikaz [;] [ ... n ]
[ END ]}[;]
```

Výpis 16: Syntaxe – Modifikace procedury

Smazání procedury Příkazem `DROP PROCEDURE` vymažeme uloženou proceduru. Za tento příkaz je nutno uvést název procedury. Syntaxi pro tento příkaz můžeme vidět na výpisu 17.

```
DROP PROCEDURE [ nazev_databaze . [ nazev_schematu ]. ] nazev_procedury [ ,...n ] [ ; ]
```

Výpis 17: Syntaxe – Smazání procedury

3.4.6 Funkce

Vytvoření skalární funkce Skalární funkce tvoříme dle syntaxe ve výpisu 18. Za klíčová slova `CREATE FUNCTION` uvedeme název funkce, volitelně název schématu. Do závorky poté uvedeme vstupní parametry, kdy se před každý parametr zapíše znak `@`. Za každý z těchto parametrů se poté uvede datový typ (název a datový typ můžeme oddělit klíčovým slovem `AS`), které chceme funkci předávat. Za tyto parametry uvedeme klíčové slovo `RETURNS`, za které definujeme jaký datový typ bude funkce vracet. Poté opět můžeme uvést klíčové slovo `AS` a za něj již musíme uvést slovo `BEGIN` (na rozdíl od procedury, kde je použití `BEGIN -- END` volitelné). Tato slova nám uvozují samotné tělo funkce, ve kterém je třeba definovat návratovou hodnotu za klíčové slovo `RETURN`, které musí být vždy posledním příkazem ve skalární funkci. Celý tento blok je ukončen výrazem `END`.

```
CREATE FUNCTION [ nazev_schematu. ] nazev_funkce
( [ { @parametr [ AS ] datovy_typ_parametru
  } [ ,... n ]
  ]
)
RETURNS navratovy_datovy_typ
    [ AS ]
    BEGIN
        telo_funkce
        RETURN skalarni_vyraz
    END [ ; ]
```

Výpis 18: Syntaxe – Vytvoření skalární funkce

Modifikace skalární funkce Modifikace těchto funkcí se provádí podobně jako vytváření skalární funkce. Zaměníme pouze klíčové slovo `CREATE` před slovem `FUNCTION` za slovo `ALTER`. Dostaneme tedy syntaxi uvedenou ve výpisu 19. Příkaz `ALTER` lze provést pouze v případě, pokud zachováme parametry. Pokud by se počet parametrů změnil, bylo by potřeba provést `DROP` a `CREATE`. Toto platí i pro procedury.

```
CREATE FUNCTION [ nazev_schematu. ] nazev_funkce
( [ { @parametr [ AS ] datovy_typ_parametru
  } [ ,... n ]
  ]
)
RETURNS navratovy_datovy_typ
    [ AS ]
    BEGIN
        telo_funkce
        RETURN skalarni_vyraz
    END [ ; ]
```

Výpis 19: Syntaxe – Modifikace skalární funkce

Vytvoření Inline Table – Valued funkce Můžeme si všimnout, že popis syntaxe je velice podobný jako v případě pro vytvoření skalární funkce. Mění se pouze druhá část této syntaxe. Místo datového typu uvedeného za klíčovým slovem `RETURN` uvedeme klíčové slovo `TABLE`, které nám říká, že funkce vrací tabulku, která je získána nějakým `SELECT` příkazem.

```
CREATE FUNCTION [ nazev_schematu. ] nazev_funkce
( [ { @parametr [ AS ] datovy_typ_parametru
  } [  ,... n ]
  ]
)
RETURNS TABLE
    [ AS ]
    RETURN [ ( ] prikazy_select [ ) ]
    END [ ; ]
```

Výpis 20: Syntaxe – Vytvoření Table – Valued funkce

Modifikace Inline Table – Valued funkce Modifikace těchto funkcí probíhá podle podobného popisu syntaxe jako v předchozím případě. Musí se zaměnit pouze klíčové slovo `CREATE` za slovo `ALTER`. Výsledná syntaxe je uvedena na výpisu 21.

```
ALTER FUNCTION [ nazev_schematu. ] nazev_funkce
( [ { @parametr [ AS ] datovy_typ_parametru
  } [  ,... n ]
  ]
)
RETURNS TABLE
    [ AS ]
    RETURN [ ( ] prikazy_select [ ) ]
    END [ ; ]
```

Výpis 21: Syntaxe – Modifikace Inline Table – Valued funkce

Vytvoření Multistatement Table – Valued funkce Syntaxe vytvoření této funkce je uvedena na výpisu 22. Můžeme vidět, že první část syntaxe je stejná jako v předchozím případě. Mění se druhá část, která začíná klíčovým slovem `RETURN`, za které se deklaruje proměnná tabulky, která se bude vracet. Poté se definuje samotná tabulka, za kterou se může uvést klíčové slovo `AS`. Mezi klíčová slova `BEGIN a END` se poté zapisuje samotné tělo funkce, kde se naplní návratová proměnná. Tělo funkce končí klíčovým slovem `RETURN`.

```
CREATE FUNCTION [ nazev_schematu. ] nazev_funkce
( [ { @parametr [ AS ] datovy_typ_parametru
  } [  ,... n ]
  ]
```

```
)
RETURNS @navratova_promenna TABLE <definice_tabulky>
    [ AS ]
    BEGIN
      telo_funkce
      RETURN
    END [ ; ]
```

Výpis 22: Syntaxe – Vytvoření Multistatement Table – Valued funkce

Modifikace Multistatement Table – Valued funkce Modifikace těchto funkcí se provádí podobně jako vytváření, zamění se pouze klíčové slovo `CREATE` za `ALTER`, což můžeme vidět na syntaxi uvedenou na výpisu 23. Opět je potřeba dodržet počet parametrů.

```
ALTER FUNCTION [ nazev_schematu. ] nazev_funkce
( [ { @parametr [ AS ] datovy_typ_parametru
  } [   ,... n ]
  ]
)
RETURNS @navratova_promenna TABLE <definice_tabulky>
    [ AS ]
    BEGIN
      telo_funkce
      RETURN
    END [ ; ]
```

Výpis 23: Syntaxe – Modifikace Multistatement Table – Valued funkce

Smazání funkcí Všechny funkce se smažou stejným příkazem, nezávisle na typu funkce. Tento příkaz je `DROP FUNCTION`. Popis syntaxe tohoto příkazu můžeme vidět ve výpisu 24.

```
DROP FUNCTION [ nazev_databaze . [ nazev_schematu ]. ] nazev_funkce [ ,...n ] [ ; ]
```

Výpis 24: Syntaxe – Smazání funkce

4 Přístup k systémovému katalogu

4.1 Definice systémového katalogu

Systémový katalog je kolekce pohledů, které popisují strukturu databáze. Strukturou databáze se rozumí seznam všech databázových objektů např. tabulek, pohledů a triggerů. V literatuře je taktéž nazýván pojmem „metadata“, neboli „data o datech“. V těchto objektech jsou obsaženy veškeré informace o dané databázi.

4.2 Přístup k systémovému katalogu

Prakticky máme tyto možnosti jak přistupovat k systémovému katalogu:

- Systémové katalogové pohledy (Catalog Views)
- Pohledy na informační schéma (Information Schema Views)
- ODBC katalogové funkce (ODBC Catalog Functions)
- OLE DB řádkové sady schémat (OLE DB Schema Rowsets)
- Systémové uložené procedury a funkce (System Stored Procedures and Functions)

V této práci se budeme zabývat především systémovými pohledy (dále jen pohledy), konkrétně objektovými katalogovými pohledy a popisem jejich důležitých atributů. Tuto metodu přístupu k metadatům doporučuje i samotný Microsoft [2] a to z následujících důvodů:

- Všechna metadata jsou k dispozici jako katalogové pohledy.
- Katalogové pohledy představují metadata ve formátu, který je nezávislý na implementaci jakékoli katalogové tabulky. Z tohoto důvodu nejsou pohledy ovlivňovány změnami katalogových tabulek.
- Jména katalogových pohledů a jejich sloupců jsou samopopisná. Výsledek dotazu odpovídá tomu, co může očekávat uživatel, který má alespoň základní znalosti o funkci, na kterou se dotazuje.

Jako příklad můžeme uvést následující dotaz, který používá pohled `sys.object`. Výsledkem dotazu budou všechny objekty v databázi, které byly změněny v posledních 10-ti dnech.

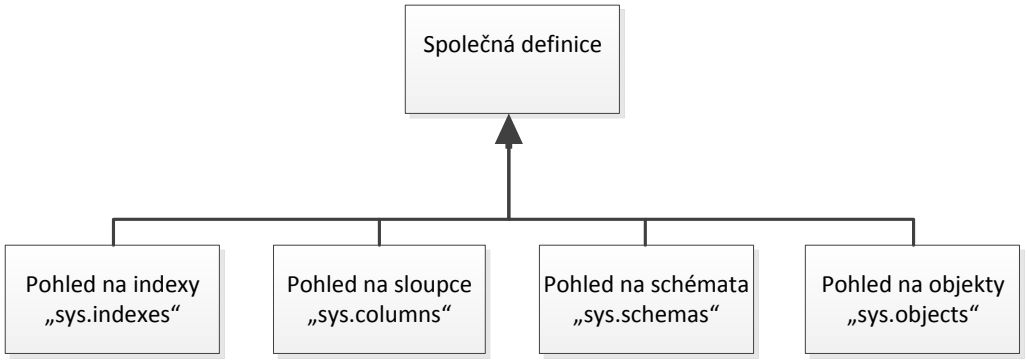
```
SELECT name AS object_name, SCHEMA_NAME(schema_id) AS schema_name,type_desc,
    create_date, modify_date
FROM sys.objects
```

```
WHERE modify_date > GETDATE() − 10
ORDER BY modify_date;
```

Výpis 25: Ukázka použití systémových pohledů.

4.3 Popis systémových pohledů

V jednotlivých částech této podkapitoly si popíšeme jak vypadá struktura systémových pohledů v databázi. V modelu na obrázku 1 můžeme vidět, že všechny tyto pohledy mají společnou definici, v tomto případě se jedná o jméno objektu. V následující části této podkapitoly si detailně popíšeme jednotlivé pohledy.



Obrázek 1: Objektový model systémových pohledů – 1. část

4.4 Pohled na indexy

Pohled obsahuje seznam všech indexů týkajících se objektů jako jsou tabulky, pohledy nebo funkce. K indexům se dostaneme pomocí systémového pohledu `sys.indexes`, který neobsahuje indexované sloupce. Tyto můžeme získat přes pohled `sys.index_columns`.

Název sloupce	Datový typ	Popis
name	sysname	Název objektu
object_id	int	Identifikační číslo objektu, kterému index náleží
name	sysname	Jméno indexu, které je jedinečné pouze v rámci objektu.
index_id	int	Identifikační číslo objektu, které je jedinečné pouze v rámci objektu.

Tabulka 4: Důležité atributy – indexy

4.5 Pohled na sloupce

Pohled umožňuje náhled na seznam všech sloupců objektů, jako jsou tabulky nebo pohledy, k nimž se dostaneme systémovým pohledem `sys.columns`. Následující seznam zahrnuje objekty, které mají sloupce:

- Vnitřní tabulky (IT)
- Systémové tabulky (S)
- SQL funkce (TF)
- Uživatelské tabulky (U)
- Pohledy (V)

Název sloupce	Datový typ	Popis
name	sysname	Název objektu
object_id	int	Identifikační číslo objektu, do kterého tento sloupec patří.
name	sysname	Jméno sloupce, které je v rámci objektu jedinečné.
column_id	int	Identifikační číslo sloupce, které je v rámci objektu jedinečné, představuje pořadí sloupce v rámci objektu.
max_length	smallint	Maximální velikost sloupce v bajtech.

Tabulka 5: Důležité atributy – sloupce

4.6 Pohled na schémata

Pohled obsahuje seznam všech všech schémat pro danou databázi. Přístup ke schématům je zajištěn pohledem `sys.schemas`.

Název sloupce	Datový typ	Popis
name	sysname	Název objektu
schema_id	int	ID schématu, které je jedinečné v databázi

Tabulka 6: Důležité atributy – schémata

4.7 Pohled na objekty

Pohled vrací seznam všech uživatelem vytvořených objektů v databázi, kromě DDL triggerů, kdy jeden navrácený řádek odpovídá jednomu objektu. K objektům se dostaneme pohledem `sys.objects`. Další možností je využít pohledu `sys.all_objects`. Tento pohled vrací všechny objekty, a to jak objekty vytvořené uživatelem, tak objekty systémové.

Název sloupce	Datový typ	Popis
name	sysname	Název objektu
object_id	int	Identifikační číslo objektu, pro každý objekt jedinečné.
schema_id	int	Identifikační číslo schématu(databáze), v kterém se objekt nachází.
parent_object	int	Identifikační číslo objektu, do kterého daný objekt patří.
type	char(2)	Typ objektu.

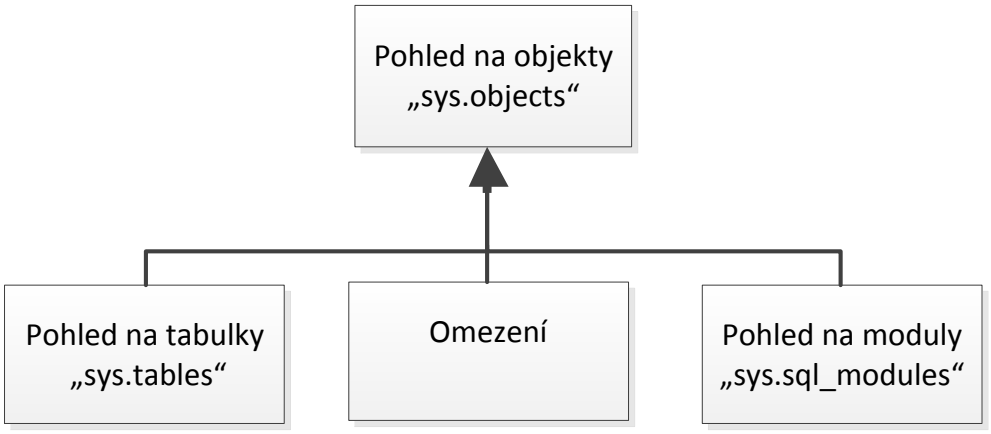
Tabulka 7: Důležité atributy – objekty

Typ objektu (sloupec type) může být nastaven na jednu z následujících možností:

- Agregační funkce (AF)
- Kontrolní omezení (C)
- Výchozí omezení (D)
- Cizí klíč (F)
- SQL skalární funkce (FN)

- Assembly scalar - funkce (FS)
- Assembly table - valued funkce (FT)
- SQL inline table-valued funkce (IF)
- Vnitřní tabulka (IT)
- Procedura (P)
- Primární klíč (PK)
- Pravidlo (R)
- Systémová bázová tabulka (S)
- Synonymum (SN)
- Sekvence (SO)
- Pohled (V)

Podrobnější popis uvedených typů naleznete níže. U těchto objektů již nebudeme uvádět parametry, které jsou shodné s `sys.object`, ale pouze důležité rozdílné parametry. K jednotlivým výše uvedeným typům se dostaneme tak, že do podmínky SQL dotazu uvedeme výraz, který je uveden u popisu každého typu.
Objekty se tedy dále dělí do těchto podskupin:



Obrázek 2: Objektový model systémových pohledů – 2. část

4.7.1 Pohled na uživatelem definované tabulky

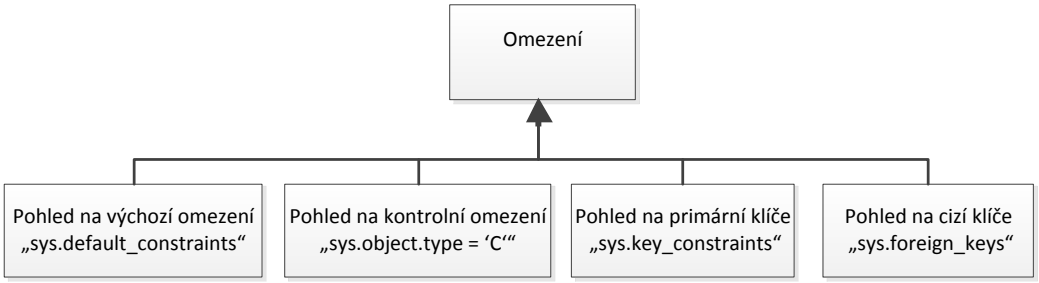
Každý záznam odpovídá jedné tabulce, která byla definována uživatelem. K těmto datům se dostaneme tak, že do podmínky výběru z pohledu `sys.objects` uvedeme tento výraz `sys.object.type = 'U'`.

Název sloupce	Datový typ	Popis
unique_index_id	int	ID odpovídající jedinečnému indexu

Tabulka 8: Důležité atributy – uživatelské tabulky

4.7.2 Omezení

Omezení odpovídají následující části objektového modelu:



Obrázek 3: Objektový model systémových pohledů – 3. část

Pohled na výchozí omezení K datům z toho pohledu lze přistupovat pomocí pohledu `sys.default_constraints`. Obsahuje řádek pro každé výchozí omezení, které je vytvořeno pomocí příkazu `CREATE TABLE` nebo `ALTER TABLE`, namísto příkazu `CREATE DEFAULT`.

Název sloupce	Datový typ	Popis
parent_column_id	int	ID sloupce v parent_object_id, kterému toto výchozí omezení náleží.
definition	nvarchar(max)	SQL výraz, který definuje tuto kontrolu omezení.

Tabulka 9: Důležité atributy – výchozí omezení

Pohled na kontrolní omezení Obsahuje řádek pro každý objekt, jehož parametr 'type' je roven 'C'. Přistupovat k těmto řádkům lze přes pohled `sys.check_constraints`.

Název sloupce	Datový typ	Popis
is_disabled	bit	Pokud je parametr roven 1, je kontrola omezení vypnuta.
definition	nvarchar(max)	SQL výraz, který definuje tuto kontrolu omezení.

Tabulka 10: Důležité atributy – kontrolní omezení

Pohled na primární klíče Každý řádek odpovídá primárnímu klíči nebo jedinečnému omezení. Potřebné informace získáme ze systémového pohledu `sys.object` s filtrem záznamů na `sys.object.type = 'PK'`. Je vidět, že primární klíč je definován indexem. To znamená, že každý primární klíč musí být zároveň indexem. Ke stejným datům se taktéž můžeme dostat přes pohled `sys.key_constraints`.

Název sloupce	Datový typ	Popis
unique_index_id	int	ID odpovídající jedinečnému indexu

Tabulka 11: Důležité atributy – primární klíče

Pohled na cizí klíče Každý řádek odpovídá objektu, který je cizí klíč. K těmto řádkům se dostaneme přes pohled `sys.foreign_keys`. Tento pohled neobsahuje záznamy se sloupci cizích klíčů. Sloupce získáme přes pohled `sys.foreign_key_columns`.

Název sloupce	Datový typ	Popis
referenced_object_id	int	Identifikační číslo referenčního objektu
key_index_id	int	Identifikační číslo klíče.

Tabulka 12: Důležité atributy – cizí klíče

4.7.3 Pohled na moduly

Pohled vrací řádek pro každý objekt, který je v SQL jazyce definován jako modul. Jedná se o objekty typu P, RF, V, TR, FN, IF, TF. Taktéž definuje objekty typu D. Seznam všech objektů viz 4.7. K těmto objektům můžeme přistupovat přes pohled `sys.sql_modules`.

Název sloupce	Datový typ	Popis
object_id	int	ID objektu v objektu.V databázi je jedinečné.
definition	nvarchar	Zdrojový kód, kterým je tento modul definován.

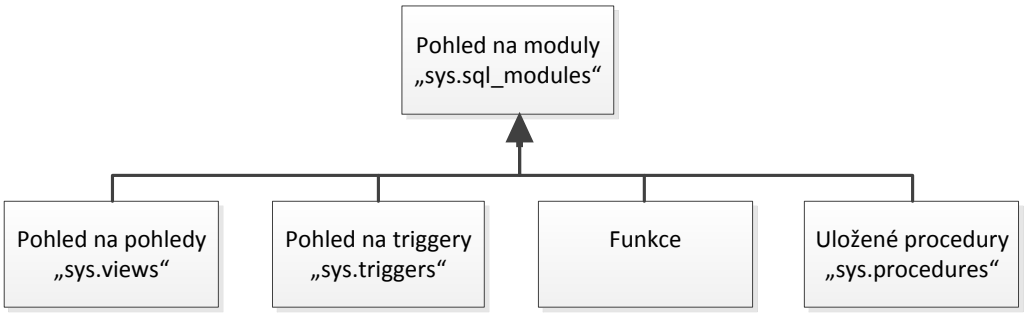
Tabulka 13: Důležité atributy – moduly

Následující ukázka kódu vrací jméno, typ a definici každého modulu v databázi. Na tento dotaz se budeme odkazovat i v dalších podkapitolách a budeme v něm měnit pouze podmínku, kterou bychom připsali do klauzule `WHERE`.

```
SELECT sm.object_id, o.name AS object_name, o.type, o.type_desc, sm.definition
FROM sys.sql_modules AS sm
JOIN sys.objects AS o ON sm.object_id = o.object_id
ORDER BY o.type;
```

Výpis 26: Ukázka výběru modulů pomocí systémových pohledů

Systémové pohledy na moduly bychom mohli vyjádřit následující objektovým modelem, jehož objekty si dále popíšeme. V každé definici dále uvedených pohledů je k dispozici vždy pouze `CREATE` (vytvoření) daného objektu tzn. pokud potřebujeme v aplikaci generovat skript pro `ALTER` je potřeba nahradit klíčové slovo `CREATE` za `ALTER`.



Obrázek 4: Objektový model systémových pohledů – 4. část

Pohled na pohledy Obsahuje řádek, pro každý objekt, jehož typ odpovídá `sys.object.type = 'V'`. K tomuto pohledu lze alternativně přistupovat tak, že data vybíráme ze systémového pohledu `sys.views`.

Pohled na triggery Každý řádek odpovídá objektu v databázi, jehož typ je nastaven na `sys.object.type = 'TR' OR type = 'TA'`. Touto podmínkou získáme všechny

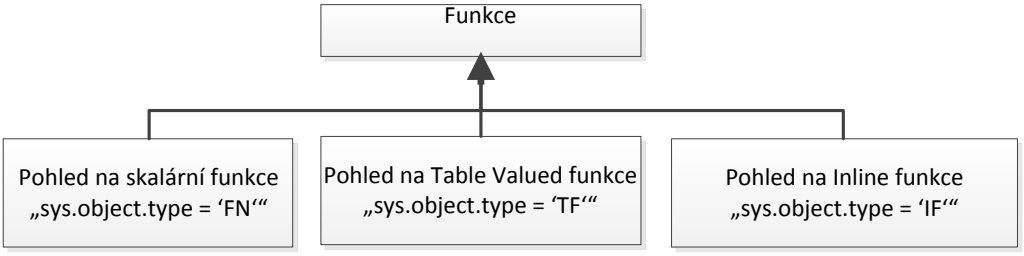
DML triggery. V případě, že chceme získat i DDL triggery je třeba triggery vybírat ze systémového pohledu `sys.triggers`.

Pohled na uložené procedury Pohled vrací řádek pro každý objekt typu „sys.object.type = 'P'". K uloženým procedurám lze taktéž přistupovat ze systémového pohledu `sys.procedures`.

Název sloupce	Datový typ	Popis
is_auto_executed	bit	1 = Procedura je vykonávána automaticky po startu serveru, v opačném případě je nastavena 0. Toto může být použito pouze pro procedury v hlavní databázi (master)

Tabulka 14: Důležité atributy – uložené procedury

Funkce Funkce v systémových pohledech odpovídají následujícímu objektovému modelu:



Obrázek 5: Objektový model systémových pohledů – 5. část

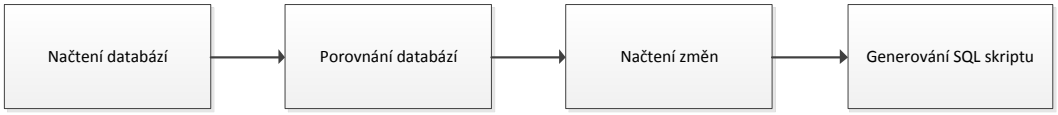
Pohled na skalární funkce Pohled na skalární funkce obsahuje řádek pro každý objekt typu `sys.object.type = 'FN'`. Tabulka s atributy viz tabulka 14.

Pohled na Table Valued funkce Pohled na Table Valued Funkce vrací řádek pro každý objekt typu `sys.object.type = 'TF'`. Tabulka s atributy viz tabulka 14.

Pohled na Inline Table – Valued funkce Tento pohled obsahuje řádek pro každý objekt typu `sys.object.type = 'IF'`. Tabulka s atributy viz tabulka 14.

5 Popis aplikace

V předchozím kapitolách jsme si popsali teoretickou část, kterou nyní využijeme při tvorbě samotné aplikace. Aplikace je vytvořena v prostředí Microsoft Visual Studio 2010 v jazyce C. Fungování aplikaci bychom mohli vyjádřit blokovým schématem uvedeného na obrázku 6. Nejprve se tedy načtou databáze, které chceme porovnat. Poté se tyto databáze porovnají, uloží se příslušné rozdíly a pro ně se poté vygeneruje SQL skript.



Obrázek 6: Fungování aplikace

5.1 Třídy pro strukturu databáze (Classes)

Ze všeho nejdříve bylo potřeba vytvořit třídy, které by odpovídaly objektům v databázi. Objekty obsažené v databázi a jejich popis můžeme nalézt v kapitole 2. Pro každý objekt je vytvořena třída, která ho reprezentuje svými vlastnostmi. Každá třída je pojmenována jako „SQLNázevObjektu“ – tedy například třída reprezentující atribut (sloupec) je pojmenována `SQLColumn`. Konkrétně třída `SQLColumn` obsahuje vlastnosti pro jméno, datový typ, informaci zda může atribut nabývat hodnot `NULL` a maximální délku atributu. Všechny dále uvedené objekty dědí z abstraktní třídy `SQLDefinition`, ve které je definována společná abstraktní vlastnost `Name`. Tato vlastnost je v potomcích přepisována.

5.2 Třídy pro práci s databází (Data)

Implementují třídu s operacemi pro každou třídu z Classes. V každé z těchto tříd je definován příkaz `SELECT` jako konstantní proměnná typu `string`, dále se zde čtou data z databáze za pomoci `SqlDataReaderu` a nakonec jsou zde implementovány metody pro navrácení kolekce objektů vytvořených ve složce Classes. Pro příklad si uvedeme třídu `DColumns`, která implementuje operace pro třídu `SQLColumn`. Ukázku pro vrácení kolekce sloupců tabulky můžeme vidět ve výpisu 27.

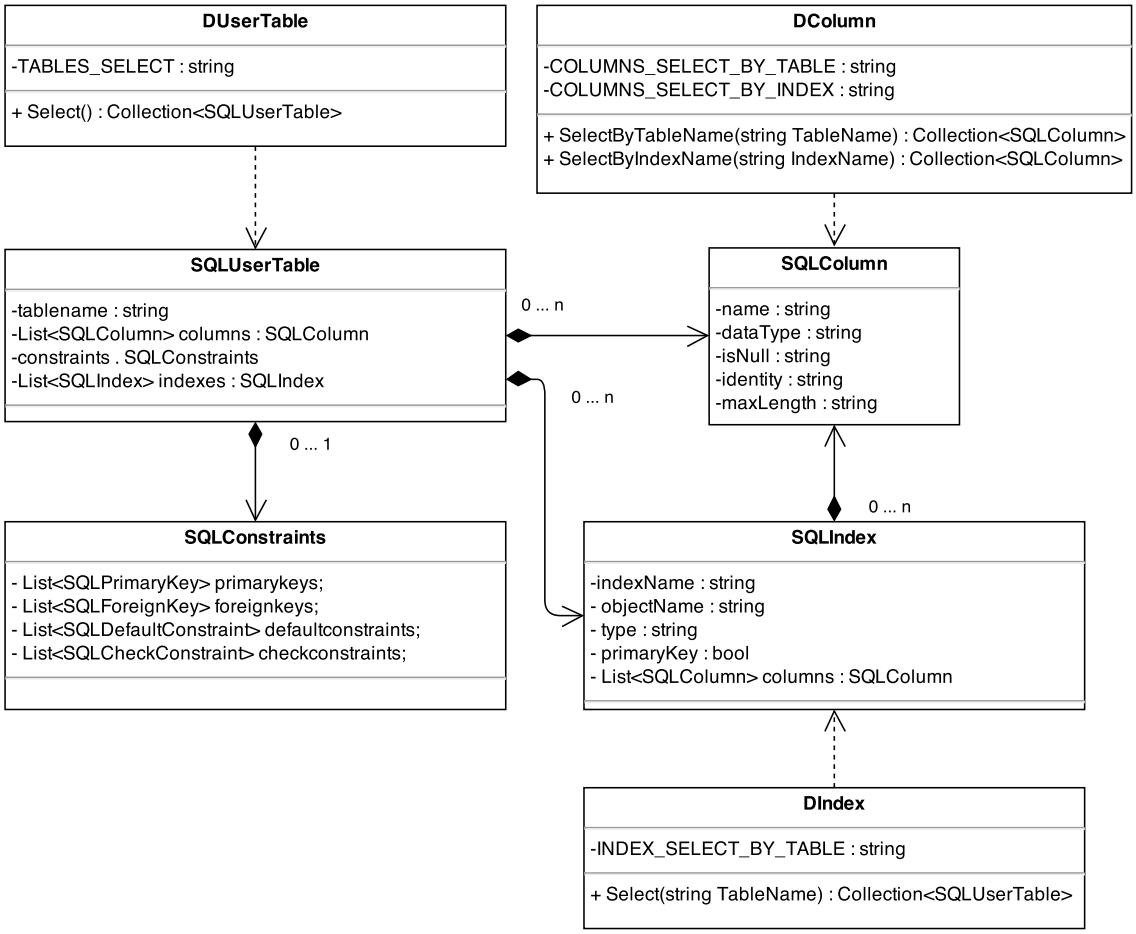
```
public Collection<SQLColumn> SelectByTableName(string TableName)
 {
                    Collection<SQLColumn> collection = new Collection<SQLColumn>();
                    Database database = new Database();
                    SqlCommand cmdSelect = database.CreateCommand(
                        COLUMNS_SELECT_TABLE);
```

```
            cmdSelect.Parameters.AddWithValue(”tableName”,TableName);
            SqlDataReader reader = cmdSelect.ExecuteReader();

            SQLColumn column = new SQLColumn();
        while (Read(reader, column))
        {
            collection .Add(column);
            column = new SQLColumn();
        }
        reader.Close();
        return collection ;
}
```

Výpis 27: Ukázka – výběr sloupců

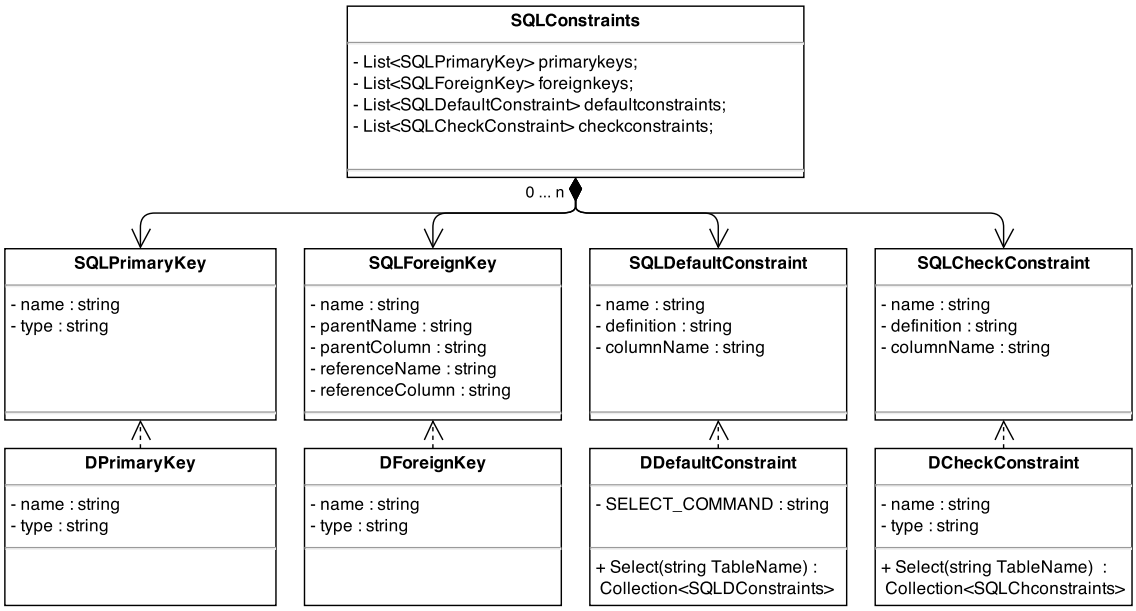
5.3 Struktura tříd

Tabulky Strukturu tříd pro tabulky můžeme vidět na obrázku 7. Jak již bylo zmíněno výše, pro každou třídu z Classes je vytvořena třída Data.



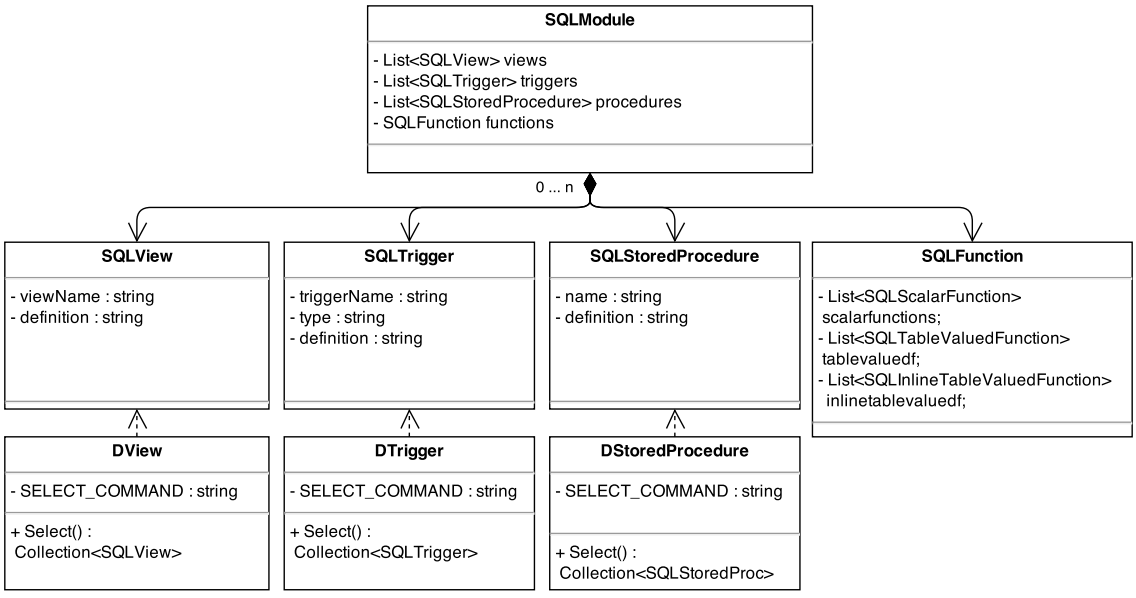
Obrázek 7: Třídní diagram – tabulky

Omezení Omezení jsou načítána pro jednotlivé tabulky, zobrazení jejich struktury je na obrázku 8.



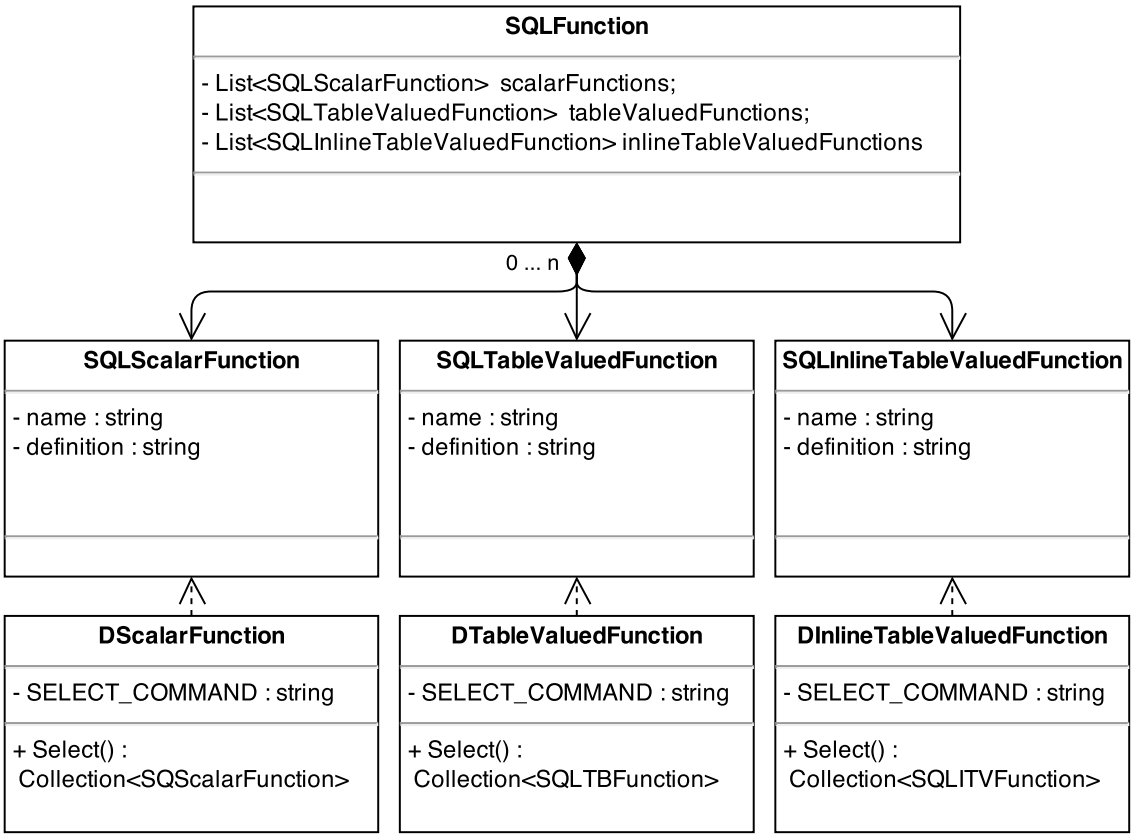
Obrázek 8: Třídní diagram – omezení

Moduly Strukturu modulů můžeme vidět na obrázku 9.



Obrázek 9: Třídní diagram – moduly

Funkce Funkce tvoří strukturu uvedenou na obrázku 10



Obrázek 10: Třídní diagram – funkce

5.4 Třídy pro provádění operací (Service)

Tyto třídy (Database, Service, Comparer a ScriptGenerator) obsahují metody pro načítání databází, jejich porovnávání a metody pro generování SQL skriptů.

Třída Database Tato třída implementuje metody pro vytvoření a ukončení připojení k databázi, dále metody pro vytvoření, potvrzení a vrácení transakce. Nakonec metodu pro vytvoření příkazu pro databázi.

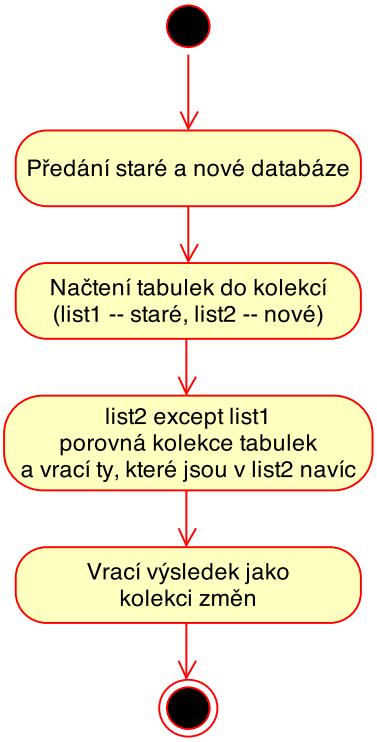
Třída Service V této třídě se nachází veřejná metoda pro načtení celé databáze, která seskupuje metody pro vytváření kolekcí jednotlivých objektů. V této třídě můžeme rovněž nalézt metody sloužící pro naplnění komponenty TreeView. Do této komponenty se zobrazují načtené databáze a také jednotlivé změny. TreeView byl použit zejména proto, že je

v něm zřetelně vidět struktura databáze. Pro příklad si uvedeme část metody pro naplnění TreeView – konkrétně část pro naplnění tabulkami se sloupci. Ukázka je uvedena ve výpisu 28. V ukázce můžeme vidět, že se nejprve vytvoří kořen stromu pro tabulky, do kterého se poté v cyklu `foreach` přidávají jednotlivé tabulky (kořeny a listy v TreeView představují tzv. „Nodes“). Pro každou tabulku se dále vytvoří list „Sloupce“, který se naplní ve vnořeném cyklu `foreach`. Podobně jako v ukázce se poté TreeView naplní zbývajícími objekty databáze, jejichž struktura je popsána v podkapitole 5.3.

```
tree.Nodes.Add("Tabulky");

foreach (SQLUserTable table in database.Tables)
{
  tree.Nodes[0].Nodes.Add(table.Name);
  tree.Nodes[0].Nodes[index].Nodes.Add("Sloupce");

  foreach (SQLColumn column in table.Columns)
    {

     tree.Nodes[0].Nodes[index].Nodes[0].Nodes.Add(column.Name + "␣" + column.DataType + "("
          + column.MaxLength + ")␣" + column.IsNull + "␣" + column.Identity);

    {
}
```

Výpis 28: Ukázka – naplnění TreeView

Třída Comparer Tato třída obsahuje funkce pro samotné porovnání objektů z načtených databází. Jako vstupní parametry těmto funkcím předáváme načtenou starou a novou databázi, ze kterých poté vybíráme objekty, které nás zajímají (tabulky, atributy, pohledy atd.). Pro každý načtený objekt z databáze jsou zde uvedeny tři funkce – jedna, která vrátí změnu jako vytvoření objektu, druhá jako smazání a třetí jako modifikaci objektu. Tyto funkce jsou typu „změna“ a vracejí kolekci změn daného typu. Třídy pro jednotlivé změny jsou popsány v podkapitole 5.5.
Pro příklad uvedeme postup při porovnání dvou tabulek, které se mají smazat. Tento příklad si znázorníme pomocí vývojového diagramu zobrazeného na obrázku 11. Funkce pro porovnání tabulek, které se mají smazat se jmenuje `CreateTables`, je typu `List<Changes.CreateTable>` a jako parametry se ji předávají stará a nová databáze.



Obrázek 11: Vývojový diagram – porovnání tabulek

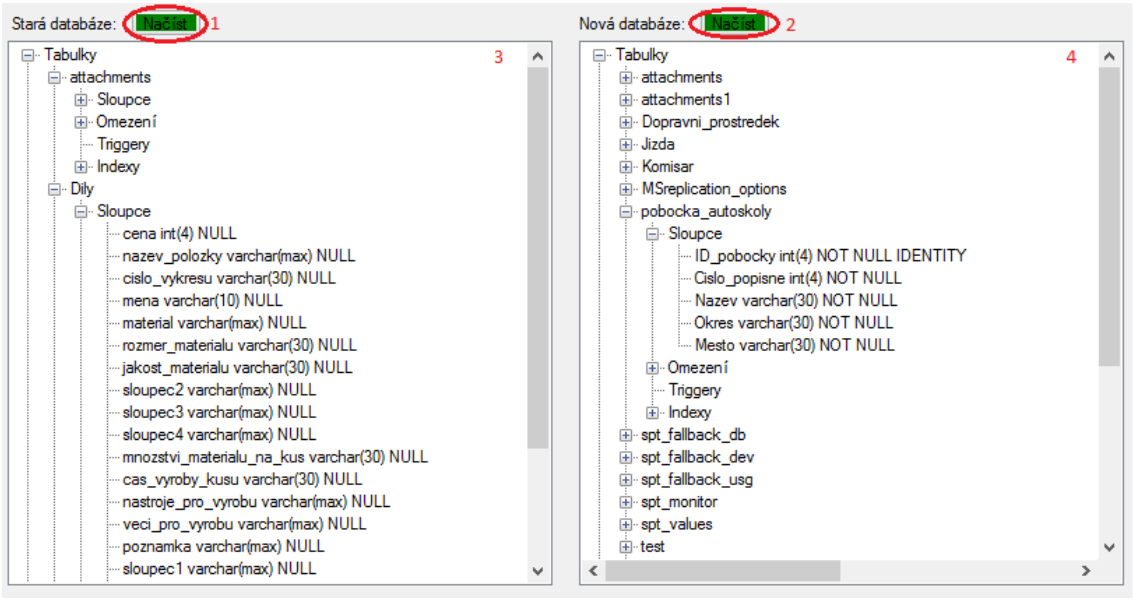
5.5 Třídy pro změny (Changes)

Složka obsahuje tři třídy pro každou ze tříd Classes 5.1. Třída `Create` obsahuje změny pro vytvoření, druhá třída `Alter` změny pro modifikaci a poslední třída `Drop` obsahuje změny pro smazání. V každé z těchto tříd je rovněž metoda pro zavolání generátoru opravného SQL skriptu. Tyto třídy jsou pojmenovány podle toho pro jakou změnu slouží. Například třída pro vytvoření pohledu se jmenuje „CreateView“.

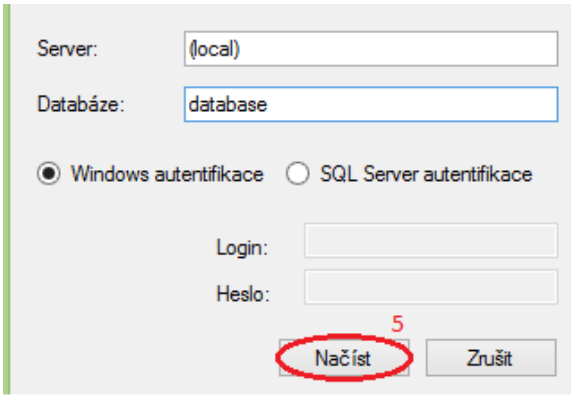
5.6 Uživatelská dokumentace

Po spuštění aplikace nás uvítá hlavní okno se třemi skupinami ovládacích prvků. První část tohoto okna je vykreslena na obrázku 12. V této části je potřeba nejprve načíst obě databáze – to se provede kliknutím na tlačítko 1 a 2. Po stisku tlačítka se zobrazí okno zobrazené na obrázku 13. V tomto okně je potřeba zadat údaje pro přístup k databázi a potvrdit je kliknutím na tlačítko 5. Jednotlivé databáze jsou poté vizualizovány pomocí stromových zobrazení 3 a 4.

Starou databázi se rozumí již používaná databáze, kterou máme nasazenou v ostrém provozu (např. ve firmě). Nová databáze je databáze, kterou jsme vytvořili jako opravu, či rozšíření staré databáze a chceme ji tedy nasadit místo původní (staré) databáze.



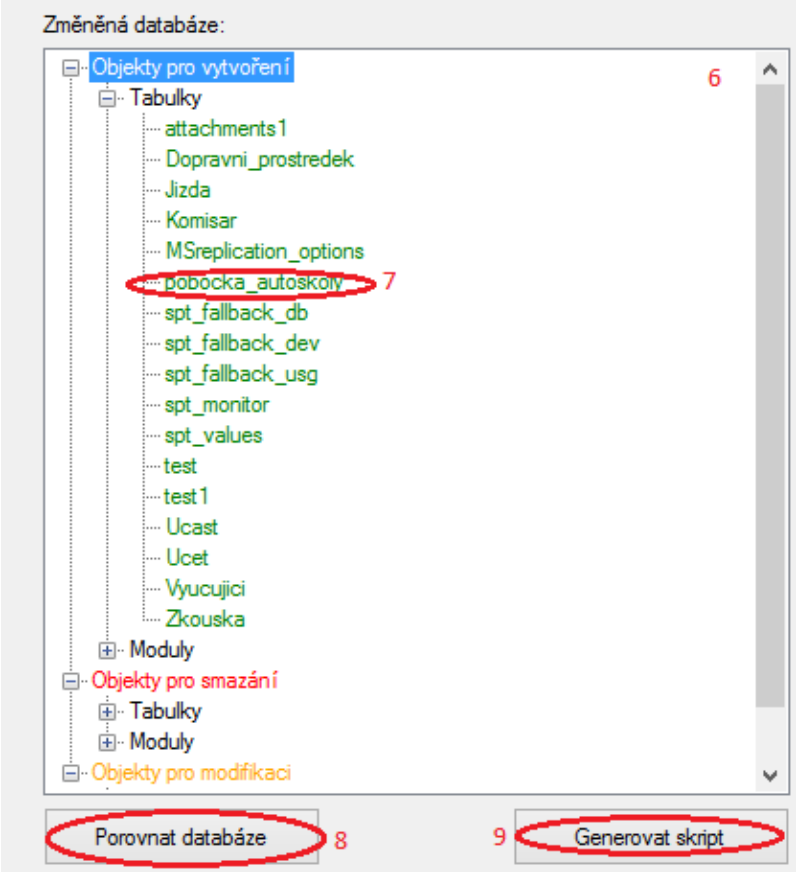
Obrázek 12: První část hlavního okna



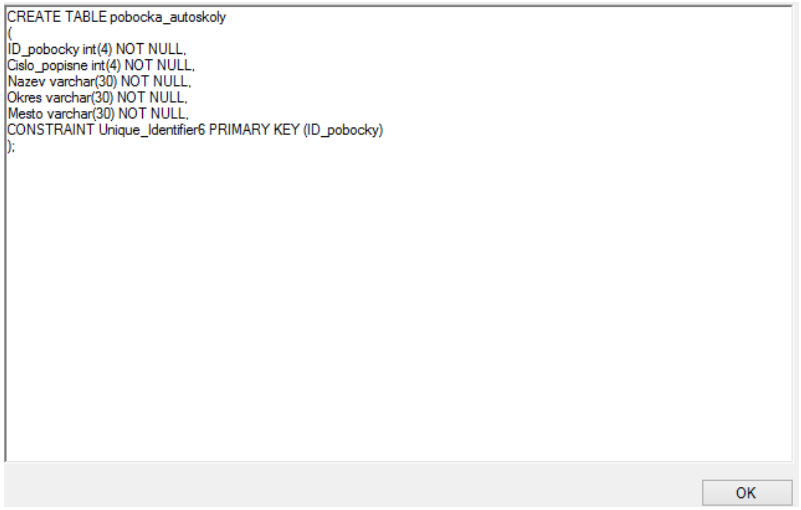
Obrázek 13: Připojení k databázi

Po načtení obou databází nás zajímá druhá část hlavního okna, která je zobrazena na obrázku 14. Zde klikneme na tlačítko 8, pomocí kterého obě načtené databáze porovnáme. Změny se vizualizují do stromového zobrazení 6. Pokud dvakrát klikneme na nějakou změnu (např. na 7 v obrázku) zobrazí se nám okno s vygenerovaným SQL skriptem

pro tuto změnu. Taktéž můžeme kliknout na tlačítko 9, po jehož stisknutí se zobrazí SQL skript pro všechny změny uvedené ve stromové struktuře 6. Příklad okna s vygenerovaným skriptem můžeme vidět na obrázku 15.



Obrázek 14: Druhá část hlavního okna



Obrázek 15: Zobrazení SQL skriptu

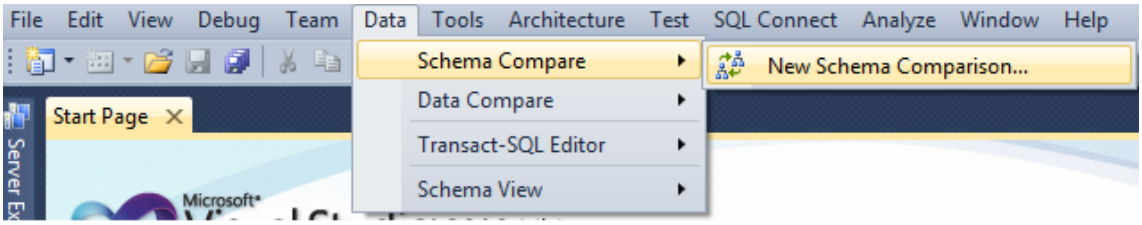
Pořadí výše uvedených úkonů je třeba dodržet (Načtení databází -> Porovnání -> Generování skriptu). Aplikace je proti jinému postupu ošetřena.

6 Srovnání s existujícími aplikacemi

Pro porovnání databází již samozřejmě existují programy, většinou tedy nejsou určeny přímo pro samotné porovnání, ale nějaká jejich část tuto funkcionalitu zvládá. Jednotlivé příklady aplikací si uvedeme v následujících podkapitolách.

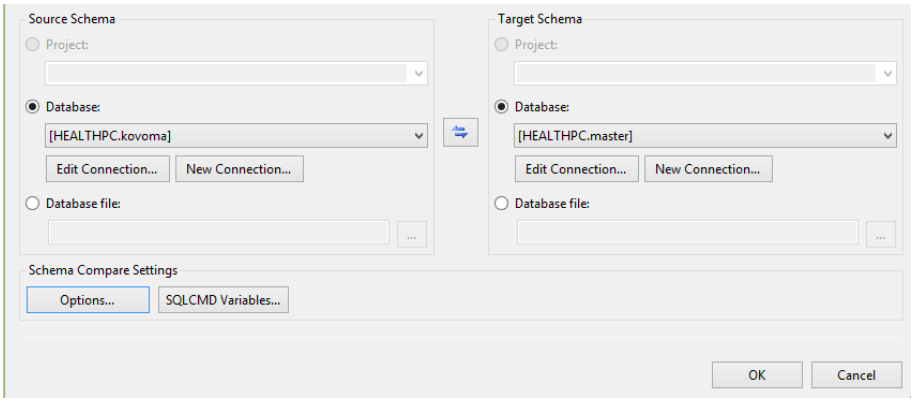
6.1 Microsoft Visual Studio

Tato aplikace nabízí porovnání databází od verze 2010, ale pouze v edicích Ultimate a Premium, verze Proffessional. Express tuto funkcionalitu nenabízí. Databáze lze v tomto programu porovnat tak, že v menu vybereme volbu Data, poté vybereme možnost Schema Compare (porovnání schémat) a klikneme na položku New Schema Comparsion (Nové porovnání schémat) – viz obrázek 16.



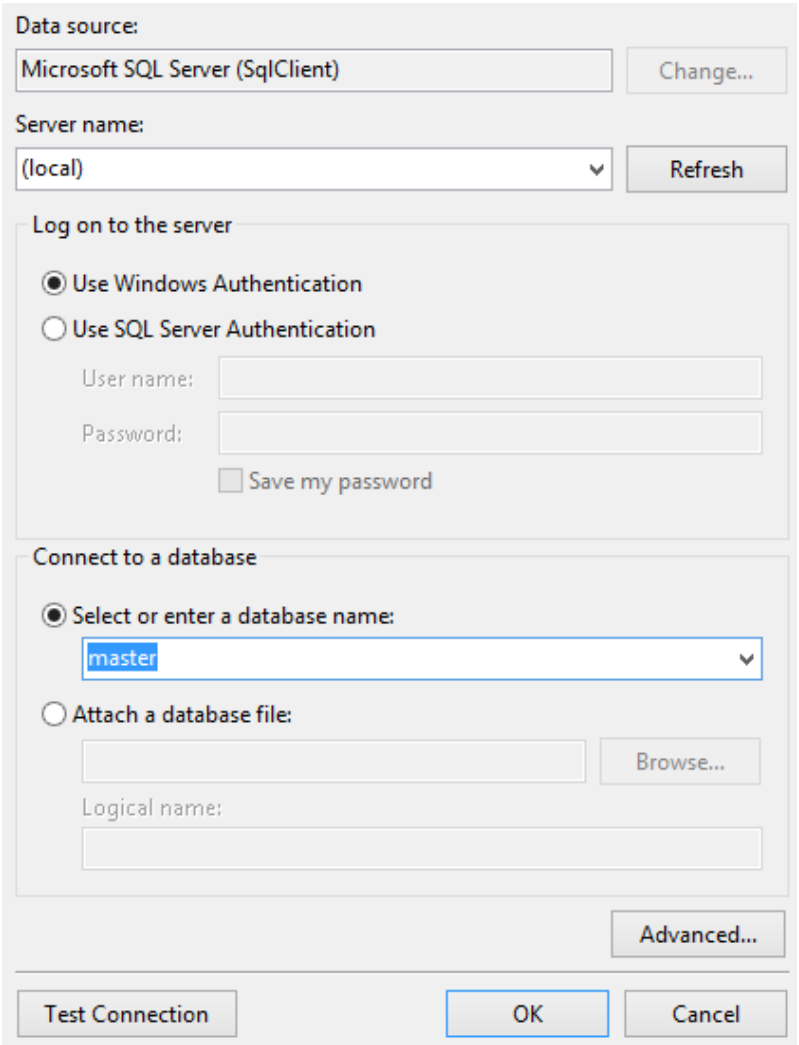
Obrázek 16: Microsoft Visual Studio – menu

Na obrázku 17 si můžeme všimnout tlačítka „Options...“. Po kliknutí na toto tlačítko se dá nastavit, které objekty se budou porovnávat. Aplikace vytvořená v této práci toto neumožňuje a porovnává všechny objekty popsané v této práci.



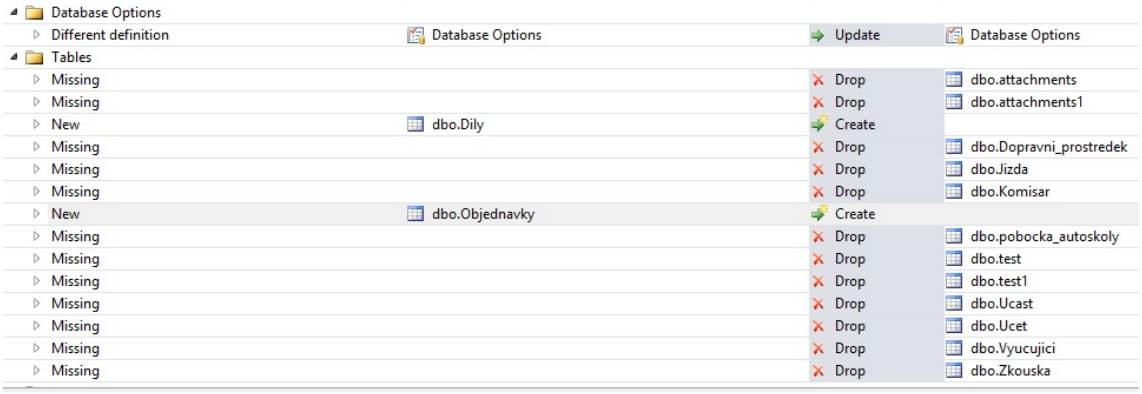
Obrázek 17: Microsoft Visual Studio – připojení

Údaje pro připojení k databázi se zadávají do okna, které se zobrazí po kliknutí na tlačítko „New Connection...“. Toto okno můžeme vidět na obrázku 18. Okno je podobné jako v naší aplikaci, ale umožňuje zvolit databázi na serveru ze seznamu, což v naší aplikaci není možné, protože musíme znát název databáze. Po stisknutí tlačítka „Test Connection“ se otestuje, zda se zle připojit k uvedené databázi. V naší aplikaci toto tlačítko není a test se provádí po stisknutí tlačítka „OK“.



Obrázek 18: Microsoft Visual Studio – údaje

Ve výsledku (obrázek 19) poté můžeme procházet jednotlivé změny a po kliknutí na změnu generujeme opravný SQL skript (obrázek 20), který se zobrazuje pod porovnání. Tato funkce porovnává mnohem více databázových objektů (např. certifikáty, SQL soubory atd...), než aplikace vytvořená v rámci této práce.



Obrázek 19: Microsoft Visual Studio – výsledek

```
CREATE TABLE [dbo].[Objednavky] (
    [ID_objednavky]     INT           IDENTITY (1, 1) NOT NULL,
    [datum_objednavky]  VARCHAR (20)  NULL,
    [cislo_objednavky]  VARCHAR (30)  NULL,
    [cislo_polozky]     VARCHAR (30)  NULL,
    [nazev_polozky]     VARCHAR (MAX) NULL,
    [cislo_vykresu]     VARCHAR (30)  NULL,
    [pocet_kusu]        INT           NULL,
    [pozadovany_termin] VARCHAR (30)  NULL,
    [dodano_kusu]       INT           NULL,
    [poznamka]          VARCHAR (MAX) NULL,
    [dodaci_list]       VARCHAR (MAX) NULL,
    [sloupec1]          VARCHAR (MAX) NULL,
```

Obrázek 20: Microsoft Visual Studio – skript

Pro srovnání výstup z námi vytvořené aplikace můžeme vidět na obrázku 14. Nový formulář s opravným SQL skriptem se zobrazí po dvojkliku na příslušnou změnu ve stromové struktuře pro změny. Výhoda naší aplikace spočívá hlavně v její jednoduchosti, což může pro menší podnik plně dostačovat, taktéž není třeba instalovat celé Visual Studio.

6.2 Další aplikace

Mezi další zástupce těchto aplikací můžeme uvést například SQL Data Compare od Red Gate Software (placený, zdarma je pouze trial verze) nebo DBComparer (zdarma). Většina těchto aplikací funguje podobně jako nástroj v Microsoft Visual Studiu – tzn. umožňuje zvolit které objekty se mají porovnávat nebo uložit aktuální projekt a poté ho načíst. Námi vytvořená aplikace toto bohužel zatím neumožňuje.

6.3 Možnosti rozšíření

Aplikace vytvořená v rámci této práce by se dala rozšířit tak, aby porovnávala více objektů nebo, aby umožňovala zvolit, které objekty se budou porovnávat – to by samozřejmě vyžadovalo popsat další objekty. Případně by se dala přidat funkce, která by výsledný opravný SQL skript přímo vykonala na staré databázi a tím ji opravila.

7 Závěr

V úvodu této práce jsme se seznámili s problematikou, kterou tato práce popisuje a zvolili jsme si, který SŘBD s objektově relačním rozšířením budeme popisovat.

V další kapitole byla rozebrána problematika příkazů jazyka T-SQL, včetně ukázek pro popis syntaxe příkazů pro definici dat, které byly později využity pro tvorbu opravných SQL skriptů v aplikaci.

Následující kapitola sloužila pro popsání objektů obsažených v databázi SQL Serveru. Tyto popsané objekty se v aplikaci načítají a poté porovnávají.

Další část této práce se zabývala popisem práce se systémovým katalogem. Zejména přístupem k tomuto katalogu v podobě systémových pohledů, které jsme si rozebrali. Pomocí těchto pohledů přistupujeme k objektům k databázi v aplikaci.

Další kapitola se věnovala především praktické části této práce, a to tvorbou aplikace pro porovnání dvou relačních databází, která pro změny generuje opravný SQL skript. V závěru této kapitoly byla aplikace porovnána s již existujícími aplikacemi. Přílohu této práce tvoří CD nosič, na kterém je umístěna samotná aplikace.
Doufám že vám tato práce pomohla k pochopení této problematiky.

Pavel Hrabovský

8 Reference

[1] Microsoft Corporation. Technet: Elektronická knihovna [online]. 2012 [cit. 2014-04-24]. Dostupné z: http://technet.microsoft.com/en-us/library/bb545450.aspx

[2] Microsoft Corporation. Technet: Elektronická knihovna, přístup k systémovým pohledům [online]. 2012 [cit. 2014-04-24]. Dostupné z: http://technet.microsoft.com/en-us/library/ms189082(v=SQL.105).aspx

[3] IBM Informix Guide To SQL [online]. 2012 [cit. 2014-04-24]. Dostupné z: http://publib.boulder.ibm.com/infocenter/idshelp/v111/index.jsp?topic=/com.ibm.sqls.doc/sqls.htm

[4] Radim BAČA, Michal KRÁTKÝ. Databázové systémy [online]. 28. září 2010. Fakulta elektrotechniky a informatiky Technická univerzita Ostrava, 2009, 38 s. [cit. 2014-04-25]. Dostupné z: http://dbedu.cs.vsb.cz/SubPages/OpenFile/OpenFile.aspx?file=dbcb/dbcb.pdf

[5] Václav KADLEC. Systémový katalog MS SQL. DB Svět [online]. [cit. 2014-04-25]. Dostupné z: http://www.dbsvet.cz/view.php?cisloclanku=2004031503

A Příloha na CD/DVD

Na přiloženém DVD se nachází tři adresáře. Adresář `SourceCode` obsahuje zdrojový kód aplikace v Microsoft Visual Studiu 2010. Adresář `TestData` obsahuje testovací data pro naplnění databází. Nakonec se zde nachází adresář `App`, ve kterém je umístěna samotná spustitelná aplikace.